3. GESTION DE PROJET

3. GESTION DE PROJET

Il n'est pas rare pour un projet logiciel de dépasser le délai estimé de 25 à 100%, c'est même à partir de ces constatations que la crise du logiciel est apparue… Aujourd'hui si quelques projets ont encore des dérapages parfois catastrophiques, certains ne dépassent les prévisions que de 5 à 10% seulement, le présent chapitre donne quelques indications pour parvenir à ce type de résultat..

Il est indispensable pour le client comme pour le fournisseur (chef de projet et management) d'estimer à l'avance la durée (calendrier) et le coût (effort) d'un projet logiciel. Il convient également d'en mesurer les risques en recensant les dépendances extérieures qui influeront sur les coûts et délais. Une remarque préalable s'impose quant au processus d'estimation : beaucoup trop souvent, le management et les clients ont de la peine à comprendre que l'estimation est une activité comportant des difficultés intrinsèques et la rendent encore plus difficile par ce manque de compréhension. Le processus requiert des raffinements successifs, il est impossible de fournir une estimation correcte jusqu'à ce que le logiciel soit appréhendé dans son intégralité.

L'estimation des coûts et durée se fait en trois étapes :

lors d'une réponse à un appel d'offres où il s'agit de fournir au plus vite une réponse adaptée au marché,

lors de la planification du projet où il s'agit d'établir le plan projet et le plan qualité qui serviront de cadre contractuel au projet,

lors du déroulement du projet afin d'affiner les prévisions et de les mettre à jour.

3.1. PROCESSUS D'ESTIMATION

La meilleure façon d'évaluer le montant à proposer dans un devis est sans doute de connaître le budget que le client est prêt à lui consacrer…Il est toutefois peu probable que le client ait la naïveté de faire part de ses prévisions de dépenses.

L'estimation du coût total d'un projet logiciel comprend le coût de développement du logiciel et du matériel, le coût de la formation, le coût des outils… Nous nous limitons ici au coût de développement du logiciel qui sera calculé en fonction du temps passé à développer celui-ci par les ingénieurs dont on connaît le tarif horaire et le taux d'implication dans le projet en terme de pourcentage de leur temps. Les coûts du développement logiciel sont donc étroitement liés à la notion de productivité des ingénieurs logiciels, c'est à dire le nombre de lignes de code (validées) produites par unité de temps. A noter qu'à ces coûts salariaux chargés, il ne faudra pas oublier de rajouter les coûts fixes selon un mode de calcul propre à chaque entreprise. Dans la suite lorsque nous parlerons de coûts, nous entendrons donc effort en terme d'hommes/mois.

Une méthode raisonnable pour établir une réponse à un appel d'offres, est de faire appel à des experts qui vont procéder par analogie avec des projets déjà achevés. On évalue chaque élément important du nouveau système en terme de pourcentage de coût d'un élément comparable dans le système achevé. L'estimation du coût total du nouveau projet est obtenue par sommation des estimations élémentaires. L'estimation est un exercice difficile, impossible en théorie et faisant donc appel à des heuristiques. Dans de trop nombreux cas, l'estimation préalable est faite à la hâte dans le but de remporter un marché, la planification qui s'ensuit est souvent pénalisée par une sous-estimation initiale des coûts et délais globaux.

Nous suggérons, en accord avec B.Boehm d'éviter de donner une estimation qui s'appuie sur un chiffre unique mais recommandons plutôt d'encadrer les prédictions par une fourchette optimiste/pessimiste. Le tableau ci dessous (Adapté *de "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0" , Bohem et al. 1995)* donne une base d'estimation pour passer d'un chiffre unique à une fourchette grâce à des coefficients multiplicateurs.

Phase	**Taille et effort**		**durée**	
	Optimiste	**Pessimiste**	**Optimiste**	**Pessimiste**
Analyse initiale des besoins	0.25	4.0	0.60	1.60
Définition approuvée des besoins	0.5	2.0	0.80	1.25
Spécification des besoins	0.67	1.5	0.85	1.15
Conception Globale	0.80	1.25	0.90	1.10
Conception détaillée	0.90	1.10	0.95	1.05

Tableau 3. 1 : Coefficients multiplicateurs pour chaque phase du projet

Pour utiliser les facteurs multiplicatifs du tableau, il suffit de multiplier le point d'estimation unique par les coefficients appropriés de manière à obtenir une fourchette d'estimation dont on peut remarquer qu'elle se

resserre au fur et à mesure que l'on avance dans les phases (0.90 à 1.10 en conception détaillée contre 0.25 à 4 en analyse initiale des besoins, donc au moment de la réponse à l'appel d'offres!)

Plutôt que d'estimer la durée du projet dans son ensemble on peut le décompose et estimer chacun de ses composants. Nous verrons que cette technique s'applique très bien à l'approche objets du fait de l'indépendance des différents objets entre eux.

Quelques règles d'or pour l'estimation

- **Eviter de deviner** : prendre le temps de réfléchir , ne jamais répondre sans avoir étudier la situation calmement
- **Prévoir du temps** pour l'estimation et la planifier
- Utiliser des données de **projets précédents**
- Prendre **l'avis des développeurs** qui vont effectivement effectuer le travail
- Estimer par **consensus** : consulter les différentes équipes , converger sur des dates au plus tôt et au plus tard
- Estimer par **difficultés** (facile, moyen, difficile)
- Ne pas oublier les **tâches récurrentes** : documentation, préparation des démonstrations utilisateurs, formation utilisateurs et développeurs, intégration à l'existant, récupération des données, revues, réunions, maintenance de l'existant pendant le développement du nouveau produit, gestion qualité, absentéisme (congés, maladie, RTT)
- Utiliser plusieurs techniques d'estimation
- Changer de méthodes d'estimation au fil de l'avancement du projet
- Prendre en compte les risques de gestion dans l'estimation

De manière générale, le processus d'estimation passe par 3 étapes

1. Estimer la taille du produit (nombre de lignes de code ou points de fonctions)
2. Estimer l'effort (en homme mois)
3. Estimer la durée (en mois ou semaines calendaires)

3.2. ESTIMATION DE LA TAILLE VIA LES POINTS DE FONCTION (ALBRECHT 79)

L'estimation des coûts passe comme nous l'avons vu en début de chapitre par l'estimation de la taille et de la productivité. Dans un premier temps nous avons défini la productivité en terme de nombre de lignes de code source produites par unité de temps, cette métrique simple présente cependant certains inconvénients:

- l'écriture de lignes de code représente une toute petite partie du développement
- pour des langages différents un même logiciel se code avec un nombre de lignes différent
- comment compter le nombre de lignes (commentaires)?
- on écrit souvent du code non livré (outils de développement)
- le nombre de lignes ne peut être compté qu'a posteriori

Un autre moyen de mesurer la productivité consiste à évaluer le nombre de fonctionnalités intéressantes produites par unités de temps. La méthode des points de fonction, définie chez IBM par Albrecht en 1979, puis rafinée en 83 propose ainsi d'estimer la taille d'un logiciel à partir de valeurs connues tôt dans le cycle de vie et qui sont indépendantes du langage choisi pour l'implémentation. La méthode est aujourd'hui défendue et mise à jour par l'IFPUG (the International Function Point User Group) qui publie the Function Point Counting Practices Manual dont la version courante est le IFPUG Manual 4.1.

Le nombre de points de fonctions d'un programme est basé sur le nombre et la complexité de cinq paramètres.:

Les données

Entrées externes (External Inputs EI) - Ces données peuvent venir d'un écran ou d'une autre application, elles peuvent être utilisées pour mettre à jour un ou plusieurs fichiers internes, il peut s'agir d'informations ou de données de contrôle. Les écrans, boites de dialogue, messages à travers lesquels un utilisateur entre une donnée font partie des entrées externes.

Sorties externes. (**External outputs EO) -** Ce sont les sorties d'un programme, elles peuvent prendre la forme de fichiers de sortie envoyés à d'autres application, d'écrans, de rapports de graphes destinés à l'utilisateur final ; ils sont créés à partir de fichiers logiques internes.

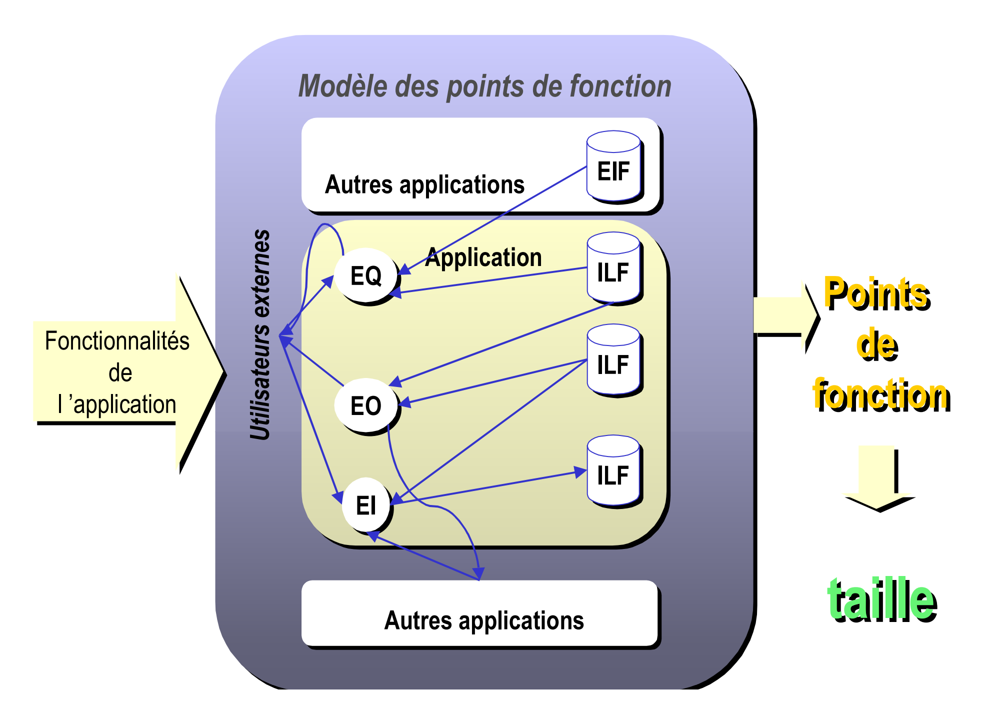


Figure 1 : Modèle des points de fonction

Les transactions

Requêtes externes (External Inquiry EQ) - Un procédé élémentaire mettant en jeu des entrées et des sorties qui résulte en la production de données provenant de plus d'un fichier logique interne. Le processus ne met pas à jour de fichier interne, combinaison d'entrées /sorties où le résultat apparaît sous forme simple et immédiate, généralement en utilisant une seule clé d'accès. La frontière entre requête externe et donnée externe est assez floue. Nous dirons que les requêtes concernent exclusivement les accès à une base de données alors que les données externes combinent ces accès avec des calculs et traitements plus ou moins complexes.

Fichiers logiques internes (**Internal Logical Files ILF's)** - Un groupe de données corrélées entièrement résidant dans les limites de l'application, mis à jour par les données externes, entièrement contrôlé par le programme. Par exemple un fichier, une table dans une base de données…

Fichier interfaces externes (External Interface Files EIF's) - Un groupe de données corrélées utilisées pour des références seulement. Les données résident entièrement hors de l'application et sont mises à jour par une autre application (c'est donc un ILF pour une autre application)

Après que les composants du projet aient été ainsi classifiés, chaque paramètre est affecté d'un score : faible, moyen, élevé. Pour les transactions (EI's, EO's, EQ's) le score est calculé sur le nombre de fichiers mis à jour ou référencés (FTR's) et le nombre de types de données impliquées (DET's).

FTR's	**Données**		
	1-4	**5-15**	**>15**
0-1	faible	faible	moyen
2	faible	moyen	haut
>2	moyen	haut	haut

Tableau 3.2: Table EI:

Une requête est notée (faible, moyenne ou haute) comme une sortie externe (EO) mais on lui attribue une valeur plutôt comme à une entrée externe. La notation est basée sur le nombre total des éléments de données et des types de fichiers référencés (combinaison "dédoublonnée" des entrées et sorties). Si un fichier ou un type est utilisé plusieurs fois il est répertorié une seule fois.

FTR's	Éléments de Données		
	1-5	**6-19**	**>19**
0-1	faible	faible	moyen
2-3	faible	moyen	haut
>3	moyen	haut	haut

Tableau 3.3 : Table EO et EQ:

Pour les fichiers externes et internes (ILF’s and EIF’s) le score (haut, moyen , élevé) est basé sur le type des éléments d' enregistrements (Record Element Type’s) et le nombre de types des données (Data Element Type’s). Un type d'élément d'enregistrement est un sous groupe reconnaissable de données dans un ILF ou EIF. Un type de donnée est un champ unique non récursif.

RET's	Data Elements		
	1-19	**20-50**	**>50**
1	faible	faible	moyen
2-5	faible	moyen	haut
>5	moyen	haut	haut

Tableau 3. 4Table ILF and EIF:

On récapitule ensuite dans un tableau les scores de chacun des paramètres et on obtient le nombre de points de fonction bruts (UFP)

Caractéristiques du programme	**Points de fonction**			
	Faible complexité	**Complexité moyenne**	**Haute complexité**	**Total**
Nombre d'entrées externes	__*3=	__*4=	__*6=	__
Nombre de sorties externes	__*4=	__*5=	__*7=	+__
Nombre de requêtes	__*3=	__*4=	__*6=	+__
Nombre de fichiers logiques internes	__*7=	__*10=	__*15=	+__
Nombre de fichiers interfaces externes	__*5=	__*7=	__*10=	+__
			Nombre total de points de fonctions bruts (UFP)	__
			Multiplié par le facteur d'ajustement* TCF	__
			Total de points de fonction ajustés AFP	__

Tableau 3.5 Multiplicateurs de points de fonction.
source: adapté de "Applied Software Measurement" (Jones 1991)

On calcule ensuite le facteur d'ajustement à partir de 14 caractéristiques générales du système. Pour chaque caractéristique on évalue le **degré d'influence** DI de 0 (aucune influence) à 5 (très grande influence). La table ci-dessous donne un aperçu des caractéristiques générales du système pour les auteurs du manuel utilisateurs de l'IFPUG.

Caractéristiques générales du système	**Description**
Communication de données	Combien de facilités de communication pour aider au transfert ou à l'échange d'information avec l'application ou le système?
Distribution du traitement et des données	Comment les données et les traitements distribués sont ils gérés
Critères de performance	L'utilisateur a t il des exigences en matière de temps de réponse?
Configuration matérielle très chargée	Quel est l'état de charge actuel de la plate-forme matérielle sur laquelle le système sera mis en exploitation?
Fréquence des transactions	Quelle est la fréquence d'exécution des transactions (quotidien, hebdomadaire, mensuel…)
Données saisies en temps réel	Quel est le pourcentage de données saisies en temps réel?
Efficacité des interfaces utilisateur	Les interfaces ont elles été dessinées pour atteindre à l'efficacité maximale de l'utilisateur
Mise à jour en temps réel des fichiers internes logiques	Combien de fichiers logiques internes sont ils mis à jour en temps réel?
Calculs complexes	L'application fait elle appel à des traitements logiques ou mathématiques complexes?
Réutilisabilité	L'application est elle développée pour satisfaire un ou plusieurs besoins clients?
Facilité d'installation	Quelle est la difficulté de conversion et d'installation?
Facilité opérationnelle	Quelle est l'efficacité et /ou l'automatisation des procédures de démarrage, backup, et récupération en cas de panne ?
Portabilité	L'application est elle spécifiquement conçue, développée maintenue pour être installée sur de multiples sites pour de multiples organisations?
Evolutivité	L'application est elle spécifiquement conçue, développée maintenue pour faciliter le changement?

Tableau 3.6 : Caractéristiques générales du système

Une fois que les caractéristiques générales du système ont été évaluées avec leur degré d'influence respectif, on peut calculer le **degré d'influence DI.** (0<DI<70) qui est la somme des degrés d'influence de chaque caractéristique.

Le facteur d'ajustement ou facteur technique de complexité se calcule par :

TCF **= 0.65 + DI/100 , on a (0.65<TCF<1.35)**

Et enfin on calcule le nombre de points de fonctions ajustés:

FP = TCF * UFC

On peut maintenant calculer les coûts, l'effort et le calendrier à partir d'expériences précédentes, ou bien utiliser une méthode rapide de calcul du planning (voir ci dessous).

De nombreuses expériences ont montré la supériorité de l'évaluation par les points de fonction (Jones constate une marge d'erreur de 200% avec les points de fonction alors qu'elle est de 800% avec le nombre de lignes de code source!!.)

La méthode des points de fonction ne repose sur aucune technologie, il y a évidemment une relation entre le nombre de lignes de codes et le nombre de points de fonction que l'on peut trouver dans les archives de sa propre entreprise ou bien en utilisant les tables de conversion nombre de points de fonction / nombre de lignes de code que l'on peut trouver sur le site de l'IFPUG (table de correspondance répertoriant plus de 500 langages). Cette table permet de prédire le nombre de lignes assez tôt. Elle permet aussi d'étudier les productivités comparées lors de programmation dans différents langages et de convertir la taille d'une application dans n'importe quel langage.

Comme on peut le voir sur les caractéristique générales retenues pour un système, la méthode des points de fonctions a été définie pour des projets essentiellement orientés gestion. En 1986 la méthode dite des Features

points propose des extensions pour pouvoir être appliquée en contexte industriel (logiciels embarqués, télécoms, systèmes d'exploitation), elle prend en compte un nouveau paramètre, le nombre d'algorithmes. Sur les projets "systèmes d'information les deux méthodes convergent mais sur un petit système temps réel on a pu constater des divergences pouvant aller jusqu'à 45 points de fonction contre 70 *features points*.

3.3. ESTIMATIONS DE L'EFFORT: MODELE COCOMO (BOEHM, 1981, 1998)

L'estimation de l'effort (en homme mois) fait suite à l'estimation de la taille (en lignes de code source) et permettra de définir un calendrier pour le projet.

La méthode COCOMO fournit un algorithme permettant de dériver une évaluation, de l'effort et du planning à partir de l'estimation de la taille du logiciel. Nous donnons en référence la méthode de (Putman and Myers 1992) qui conduit également à une évaluation de l'effort

3.3.1 Description de la méthode

La méthode COCOMO, pour COnstructive COst Model a été développé par Dr. Barry Boehm pour estimer l'effort et le temps de développement d'un produit logiciel. A l'origine elle a été construite à partir d'une analyse des données par régression pratiquée sur 63 projets logiciels (gestion et informatique industrielle) comprenant de 2000 à 100.000 lignes de code dans l'entreprise TRW (USA). COCOMO à l'avantage d'être un modèle ouvert. Les données de calibrage, les formules et tous les détails des définitions sont disponibles. La participation à son développement est encouragée.

Aujourd'hui, COCOMO II est un nouveau produit beaucoup plus adapté à l'aspect réutilisation des composants (modules existants). La version 1998 a été calibrée sur 161 points de données, en utilisant l'approche statistique Bayesienne pour combiner les données empiriques avec les avis d'experts. De plus elle peut être re-calibrée sur les données de l'entreprise. Un nouveau modèle appelé COCOTS, est en cours de développement par l'équipe de COCOMO. Ce modèle permet d'estimer les coûts et planifier des projets logiciels utilisant des composants existants. C'est encore un projet de recherche.

Pour les projets basés sur une technologie traditionnelle, le modèle original de 1981 est encore valable, d'autant plus qu'il est maintenant rodé et bien documenté.

3.3.2 COCOMO 81

Le modèle COCOMO 81 s'appuie sur une estimation de l'effort en homme*mois (HM) calculée par la formule

Effort = C*M*tailles**

C facteur de complexité

M facteur multiplicateur

Taille en milliers de lignes de code source livrées (KDSI)

s proche de 1

Hypothèses :

KDSI *livré* :

- Exclut en général les environnements de tests, les supports de développement
- Exclut les commentaires
- Inclut le shell

HOMMES MOIS (HM)

- MM = 152 Heures (Normes américaines), tient compte des vacances, arrêts maladie...

Le modèle COCOMO 81 est en fait constitué de trois modèles :

- le modèle de base
- le modèle intermédiaire
- le modèle détaillé ou expert

Le modèle de base

Le modèle de base estime l'effort (le nombre de homme mois) en fonction du nombre de milliers d'instructions source livrées(*KDSI*), de la productivité (le nombre de lignes de code par personne par mois) et d'un facteur d'échelle qui dépend du type de projet.

Les 3 types de projet identifiés sont :

- organique :
  Il est défini par une innovation minimale, une organisation simple en petites équipes expérimentées. (ex: petite gestion, système de notes dans une école , petits systèmes d'exploitation, traducteurs)

- médian (*semi-detached*) :
  Il est défini par un degré d'innovation raisonnable, (exemples: Compilateurs, Contrôle de processus simple, système bancaire interactif)
- imbriqué
  Dans ces projets le niveau d'innovation est important , l'organisation complexe, couplage fort avec beaucoup d'interactions, (exemples: Gros système d'exploitation, Systèmes transactionnels complexes, système de contrôle aérospatial..)

TYPE DE PROJET	effort en homme mois (HM)	Temps de développement
ORGANIQUE	$2.4(KDSI)^{1.05}$	$2.5(HM)^{0.38}$
MEDIAN	$3.0(KDSI)^{1.12}$	$2.5(HM)^{0.35}$
IMBRIQUE	$3.6(KDSI)^{1.20}$	$2.5(HM)^{0.32}$

Tableau 3. 7 formules d'estimation COCOMO pour le modèle de base

EXEMPLES

Effort (HM)	2KDSI	8KDSI	32KDSI	128 KDSI	512 KDSI
Organique	5	21,3	91	392	
Médian	6,5	31	146	687	3250
Imbriqué	8,3	44	230	1216	6420

Tableau 3. 8 : Effort en HOMMES MOIS (HM) en fonction de la taille et du type de projet

TDEV(mois)	2KDSI	8KDSI	32KDSI	128 KDSI	512 KDSI
Organique	4.6	8	14	24	
Médian	4.8	8,3	14	24	42
Imbriqué	4.9	8,4	14	24	41

Tableau 3.9 : Temps de développement (en mois) en fonction de la taille et du type de projet

Le temps de développement commence après les spécifications fonctionnelles et s'arrête après l'intégration

De ces chiffres on peut déduire la productivité (KDSI/HM) et le nombre moyen de personnes sur le projet (FSP=HM/TDEV).

On peut ensuite calculer la distribution de l'effort par phases (en %)

- RPD (Requirements and Preliminary Design): Conception globale et Plan d'intégration
- DD (Detail Design) : Conception détaillée
- CUT (Code and Unit Test) : Programmation et Tests unitaires
- IT (Integration and Test) : Intégration

PROJET ORGANIQUE	**2 KDSI**	**8 KDSI**	**32 KDSI**	**128 KDI**	**512 KDSI**
RPD	**16**	**16**	**16**	**16**	
DD	**26**	**25**	**24**	**23**	
CUT	**42**	**40**	**38**	**36**	
IT	**16**	**19**	**22**	**25**	
PROJET MEDIAN					
RPD	**17**	**17**	**17**	**17**	**17**
DD	**27**	**26**	**25**	**24**	**23**
CUT	**37**	**35**	**33**	**31**	**29**
IT	**19**	**22**	**25**	**28**	**31**
PROJET IMBRIQUE					
RPD	**18**	**18**	**18**	**18**	**18**
DD	**28**	**27**	**26**	**25**	**24**
CUT	**32**	**30**	**28**	**26**	**24**
IT	**22**	**25**	**28**	**31**	**34**

Tableau 3. 10 : distribution de l'effort par phases en pourcentatge

ORGANIQUE	2 KDSI	8 KDSI	32 KDSI	128 KDI	512 KDSI
RPD	19	19	19	19	
DD et CUT	63	59	55	51	
IT	18	22	26	30	
MEDIAN					
RPD	24	25	26	27	28
DD et CUT	56	52	48	44	40
IT	20	23	26	29	32
IMBRIQUE					
RPD	30	32	34	36	38
DD et CUT	48	44	40	36	32
IT	22	24	26	28	30

Tableau 3.11 distribution du temps de développement par phases en pourcentage

EXEMPLE : Soit un projet estimé à 32 *KDSI* en mode organique,

HM = $2.4 * (32)^{1.05}$ = 91 HM

TDEV = $2.5 * (91)^{0.38}$ = 14 Mois

PRODUCTIVITE = 32000 DSI/91 HM = 352 DSI/HM

FSP = 91 HM / 14 MOIS = 6.5 FSP

Conformément au tableau 3.10, on a

PHASE DE CONCEPTION : 0.16 * 91 = 14.5 HM

PHASE DE CODAGE : 0.62 * 91 = 56.5 HM

PHASE D'INTÉGRATION : 0.22 * 91 = 20 HM

Conformément au tableau 3.11, on a :

PHASE DE CONCEPTION : 0.19 * 14 = 2.6 Mois

PHASE DE CODAGE : 0.55 * 14 = 7.7 Mois

PHASE D'INTÉGRATION : 0.26 * 14 = 3.7 Mois

En divisant on obtient le nombre de personnes nécessaires pour chaque phase.

Le modèle intermédiaire

Le modèle de base ne prend en compte que le nombre de lignes source et induit des discontinuités un peu brutales au niveau du personnel entre chaque phase du cycle de vie ; ce qui peut perturber l'organisation du projet

Le modèle intermédiaire introduit 15 facteurs de productivité (appelés *'cost drivers'*), représentants un avis subjectif du produit, du matériel, du personnel, et des attributs du projet. Chaque facteur prend une valeur nominative de 1, et peut varier selon son importance dans le projet. Ces facteurs sont à rapprocher des caractéristiques générales du produit vus plus haut dans le paragraphe sur les points de fonction. Les 15 facteurs sont multipliés entre eux pour donner un facteur d'ajustement qui vient modifier l'estimation donnée par la formule de base.

Facteurs de productivité

- Logiciel
  - *RELY*: Fiabilité requise
  - *DATA*: Volume des données manipulées
  - *CPLX*: Complexité du produit
- Matériel
  - *TIME*: Contraintes de temps d'exécution
  - *STOR*: Contraintes de taille mémoire
  - *VIRT*: Instabilité de la mémoire
- Personnel
  - *ECAP*: Aptitude de l'équipe
  - *AEXP*: Expérience du domaine
  - *VEXP*: Expérience de la machine virtuelle
  - *LEXP*: Maîtrise du langage
- Projet

- *MODP*: Pratique de développement évoluées
- *TOOL*: Utilisation d'outils logiciels
- *SCED*: Contraintes de délais

Multiplicateurs	Très bas	Bas	Nominal	Elevé	Très élevé	Extrêmement élev
RELY	0,75	0,88	1	1,15	1,4	
DATA		0,94	1	1,08	1,16	
CPLX	0,7	0,85	1	1,15	1,3	1,65
TIME			1	1,11	1,3	1,66
STOR			1	1,06	1,21	1,56
VIRT		0,87	1	1,15	1,3	
ECAP	1,44	1,18	1	0,86	0,7	
AEXP	1,29	1,13	1	0,91	0,82	
VEXP	1,21	1,1	1	0,9		
LEXP	1,14	1,07	1	0,95		
MODP	1,24	1,1	1	0,91	0,82	
TOOL	1,24	1,1	1	0,91	0,83	
SCED	1,23	1,08	1	1,04	1,1	

Tableau 3.12 Coefficients multiplicateurs pour chacun des facteurs de productivité

TYPE DE PROJET	effort en homme mois (HM)	Temps de développement
ORGANIQUE	$3.2(KDSI)^{1.05}$	$2.5(HM)^{0.38}$
MEDIAN	$3.0(KDSI)^{1.12}$	$2.5(HM)^{0.35}$
IMBRIQUE	$2.8(KDSI)^{1.20}$	$2.5(HM)^{0.32}$

Tableau 3. 13 : coefficients pour le calcul d'effort et de temps dans le modèle intermédiaire

Calcul de l'effort (HM) dans le modèle intermédiaire

Identifier le mode de développement - organique, médian, imbriqué) . Ceci donne 4 coefficients au modèle : p (productivité nominative), e (échelle appliquée à la taille du logiciel), c (constante du modèle) et d (échelle appliquée au temps de développement) conformément au tableau 3.13.

Estimer le nombre de lignes du code source (en KDSI), puis calculer le nombre d'homme*mois par la formule appropriée du tableau 3.13. On obtient HM base

Estimer les 15 facteurs de productivité et calculer le facteur d'ajustement (a) en multipliant les facteurs ensemble conformément au tableau 3.12

Multiplier l'effort 'nominal' par le facteur d'ajustement : HM = HM base * a

Calculer le temps de développement en utilisant le tableau 3.13: TDEV = c(HH)^d (on notera que les coefficients restent inchangés par rapport au modèle de base)

Dans le modèle intermédiaire, les coefficients multiplicateurs s'appliquent uniformément sur l'ensemble des phases.

Modèle expert

Le modèle expert inclut toutes les caractéristiques du modèle intermédiaire avec une estimation de l'impact de la conduite des coûts sur chaque étape du cycle de développement : définition initiale du produit, définition détaillée, codage, intégration . De plus, le projet est analysé en terme d'une hiérarchie : module, sous système et système. Il permet une véritable gestion de projet, utile pour de grands projets. Le modèle expert n'est pas décrit ici.

3.3.2 COCOMO II

A la lecture du paragraphe précédent sur COCOMO 81, plusieurs questions viennent à l'esprit:

- une seule entreprise est-elle représentative comme base de développement de COCOMO?
- COCOMO reste très lié au nombre de lignes de code, surtout le modèle de base, mais plus les programmeurs sont experts (et leur salaire élevé), moins ils écrivent de lignes de code pour un même projet!
- Comment prendre en compte le réutilisation?

Afin de prendre en compte ces critiques, COCOMO II (Boehm 98)est apparue.

COCOMO II peut être calibré pour mieux correspondre aux projets de l'entreprise. COCOMO II tient compte de la réutilisation

COCOMO II est constitué de trois modèles :

Modèle de composition d'application :

Ce modèle est utilisé pour les projets fabriqués à l'aide des toolkits d'outils graphiques. Il est basé sur les nouveaux 'Object Points'.

Modèle avant projet :

Modèle utilisé pour obtenir une estimation approximative avant de connaître l'architecture définitive. Il utilise un sous ensemble de facteurs de productivité (cost drivers). Il est basé sur le nombre de lignes de code ou les points de fonction non ajustés.

Modèle post-architecture :

Il s'agit du modèle le plus détaillé de COCOMO II. A utiliser après le développement de l'architecture générale du projet. Il utilise des facteurs de productivité (cost drivers) et des formules.

Les facteurs utilisés sont classés en : Facteurs d'échelle, Urgence, Flexibilité de développement, résolution d'architecture/Risque, Cohésion d'équipe et Maturité de Processus.

3.3.3 Conclusion

COCOMO reste le plus connu des modèles d'estimation de coût de projet logiciel. Mais le modèle d' origine n'adresse pas les projets utilisant les composants logiciel existants. COCOMO II répond à ce problème, mais est encore jeune, Le projet de recherche appelé COCOTS est à suivre pour les personnes désirant estimer un projet logiciel moderne.

COCOMO reste la référence en matière d'estimation détaillée des coûts et surtout de la ventilation de ces coûts suivant les phases des projets. Les estimations de COCOMO sont d'autant plus fiables, que les paramètres du projet sont bien connus, c'est-à-dire qu'on a profité des projets précédents pour étalonner ces paramètres.

La principale faiblesse réside dans la nécessité d'avoir une estimation du nombre de lignes du logiciel. Cette taille du logiciel n'est connue qu'à la fin de la réalisation et le problème de son estimation reste entier même si l'approche par les points de fonction peut fournir une aide. Ainsi, le management devra évaluer la justesse des prévisions tout au long de la durée de vie du projet.

Les résultats montrent que les valeurs réelles sont égales à 20% près aux valeurs estimées par COCOMO dans 68 % des cas .

3.4. AUTRES METHODES D'ESTIMATION

3.4.1 Estimation rapide du calendrier (Jone's First-Order Estimation Practice)

Une méthode d'estimation rapide du temps de développement consiste à le calculer à partir de l'effort en utilisant la formule:

Temps de développement en mois = 3.0 * HM^(1/3) (ou : $TDEV=3.0*effort^{(1/3)}$)

Jones définit 3 types de projets

- logiciels systèmes: systèmes d'exploitation, pilotes, compilateurs, bibliothèques de code
- logiciel de gestion: systèmes d'information utilisés par une seule entreprise.
    - **Business Software**:in-house systems that are used by a single organization. They run on a limited set of hardware, perhaps only a single computer.Payroll systems, accounting systems, inventory control system, as well as (there) IS,IT and MIS software are in that category.
    - **Shrink-wrap Software**:software that is packaged and sold commercially.(word processors,spreadsheet, but also financial analysis software, screenplay-writing and legal case management programs)

Systems software does not include Embedded software,firmware,real-time sytems,scientific sofware and the like. Productivity for this kinds of systems would be much lower. For you particular project, you can mix the models , for example 40% Business, 60% shrink-wrap, and recompute the schedule and effort obtained with the following tables with these proportions.

Le tableau 3.14 emprunté à Jones donne une estimation de la durée à partir du nombre de points de fonction.

Type de logiciel	Meilleur de la classe	Moyen	Pire de la classe
Systèmes	0.43	0.45	0.48
Gestion	0.41	0.43	0.46
Petits projets	0.39	0.42	0.45

Tableau 3.14 Exposants pour le calcul de la durée à partir des points de fonction: *source: adapté de "Determining Software Schedules" (Jones 1995c).* Ces exposants ont été calculés à partir de l'analyse de milliers de projets.

Exemple : pour un petit projet ayant 350 points de fonction confié à une équipe moyenne on trouve $350^{0.42}$ soit environ 12 mois calendaires.

Cette pratique ne remplace pas des méthodes plus précises mais donne une approximation rapide qui vaut mieux que la simple devinette.

On notera que l'on retrouve la même forme de métrique que dans COCOMO 81 de base et des chiffres sensiblement équivalents .

Estimation Ballpark Schedule:

<u>Before using these tables, you may want to reduce the schedule, here is how to recompute effort (possible if you use nominal project table..):</u>

Schedule Compression factor= desired schedule/initial schedule

compressed schedule effort = initial effort/Schedule Compression factor

If you have an initial schedule of 12 months and an initial effort of 78 man months, and you want a 10 months schedule: that yield a compressed schedule effort of 94 man months which means that the 17 percent reduction in the schedule requires a 21 percent increase in effort **Most researchers have concluded that it isn't possible to achieve a schedule compression factor lower than about 0.75-0.80** (Boehm 1981; Putnam and Myers 1992, Jones 1994).

Shortest possible Schedule:it is more a theorical limit you can't reach

source: derived from data in "Software Engineering Economics"(Boehm 1981),"An Empirical Validation of Software Cost Estimation Models"(Kemerer 1987),"Applied software Measurement"(Jones 1991),"Measures for Excellence"(Putman and Myers 1992),and "Assessment and Control of Software risks"(Jones 1994).

	Systems Products		**Business Products**		**Shrink-Wrap products**	
System Size (lines of code)	**Schedule (calendar months)**	**Effort (man-months)**	**Schedule (calendar months)**	**Effort (man-months)**	**Schedule (calendar months)**	**Effort (man-months)**
10,000	6	25	3.5	5	4.2	8
15,000	7	40	4.1	8	4.9	13
20,000	8	57	4.6	11	5.6	19
25,000	9	74	5.1	15	6	24
30,000	9	110	5.5	22	7	37
35,000	10	130	5.8	26	7	44
40,000	11	170	6	34	7	57
45,000	11	195	6	39	8	66
50,000	11	230	7	46	8	79
60,000	12	285	7	57	9	98
70,000	13	350	8	71	9	120
80,000	14	410	8	83	10	140
90,000	14	480	9	96	10	170
100,000	15	540	9	110	11	190
120,000	16	680	10	140	11	240
140,000	17	820	10	160	12	280
160,000	18	960	10	190	13	335
180,000	19	1,100	11	220	13	390
200,000	20	1,250	11	250	14	440
250,000	22	1,650	13	330	15	580
300,000	24	2,100	14	420	16	725
400,000	27	2,900	15	590	19	1,000
500,000	30	3,900	17	780	20	1,400

Efficient Schedule:the one you could reach with a good team , and very good management

source: derived from data in "Software Engineering Economics"(Boehm 1981),"An Empirical Validation of Software Cost Estimation Models"(Kemerer 1987),"Applied software Measurement"(Jones 1991),"Measures for Excellence"(Putman and Myers 1992),and "Assessment and Control of Software risks"(Jones 1994).

	Systems Products		**Business Products**		**Shrink-Wrap products**	
System Size (lines of code)	**Schedule (calendar months)**	**Effort (man-months)**	**Schedule (calendar months)**	**Effort (man-months)**	**Schedule (calendar months)**	**Effort (man-months)**
10,000	8	24	4.9	5	5.9	8
15,000	10	38	5.8	8	7	12
20,000	11	54	7	11	8	18
25,000	12	70	7	14	9	23
30,000	13	97	8	20	9	32
35,000	14	120	8	24	10	39
40,000	15	140	9	30	10	49
45,000	16	170	9	34	11	57
50,000	16	190	10	40	11	67
60,000	18	240	10	49	12	83
70,000	19	290	11	61	13	100
80,000	20	345	12	71	14	120
90,000	21	400	12	82	15	140
100,000	22	450	13	93	15	160
120,000	23	560	14	115	16	195
140,000	25	670	15	140	17	235
160,000	26	709	15	160	18	280
180,000	28	910	16	190	19	320

200,000	29	1,300	17	210	20	360
250,000	32	1,300	19	280	22	470
300,000	34	1,650	20	345	24	590
400,000	38	2,350	22	490	27	830
500,000	42	3,100	25	640	29	1,100

Nominal Schedule:Average team, average project, a normal project!

source: derived from data in "Software Engineering Economics"(Boehm 1981),"An Empirical Validation of Software Cost Estimation Models"(Kemerer 1987),"Applied software Measurement"(Jones 1991),"Measures for Excellence"(Putman and Myers 1992),and "Assessment and Control of Software risks"(Jones 1994).

	Systems Products		**Business Products**		**Shrink-Wrap products**	
System Size (lines of code)	**Schedule (calendar months)**	**Effort (man-months)**	**Schedule (calendar months)**	**Effort (man-months)**	**Schedule (calendar months)**	**Effort (man-months)**
10,000	10	48	6	9	7	15
15,000	12	76	7	15	8	24
20,000	14	110	8	21	9	34
25,000	15	140	9	27	10	44
30,000	16	185	9	37	11	59
35,000	17	220	10	44	12	71
40,000	18	270	10	54	13	88
45,000	19	310	11	61	13	100
50,000	20	360	11	71	14	115
60,000	21	440	12	88	15	145
70,000	23	540	13	105	16	175
80,000	24	630	14	125	17	210
90,000	25	730	15	140	17	240
100,000	26	820	15	160	18	270
120,000	28	1,000	16	200	20	335
140,000	30	1,200	17	240	21	400
160,000	32	1,400	18	280	22	470
180,000	34	1,600	19	330	23	240
200,000	35	1,900	20	370	24	610
250,000	38	2,400	22	480	26	800
300,000	41	3,000	24	600	29	1,000
400,000	47	4,200	27	840	32	1,400
500,000	51	5,500	29	1,100	35	1,800

3.5 ÉLEMENTS DE PLANIFICATION

On cherche à :

- Définir les activités constituant le projet
- Organiser les activités dans le temps.
  Évaluer les dépendances entre activités.
- Évaluer l'effort nécessaire pour chaque activité (durée maximum et minimum).
- Affecter les personnes aux activités.

3.4.1. Décomposition structurée des activités, (WBS)

La planification commence par un recensement des tâches à réaliser. La décomposition structurée des activités (WBS Work Breakdown Structure) permet de recenser l'ensemble des activités d'un projet et de les décomposer. La décomposition apparaît sous forme arborescente

Il s'agit d'une décomposition purement statique :elle ne tient pas compte du temps, et par conséquent ne s'attache pas à l'ordonnancement des activités. Elle permet une présentation analytique: on doit décomposer

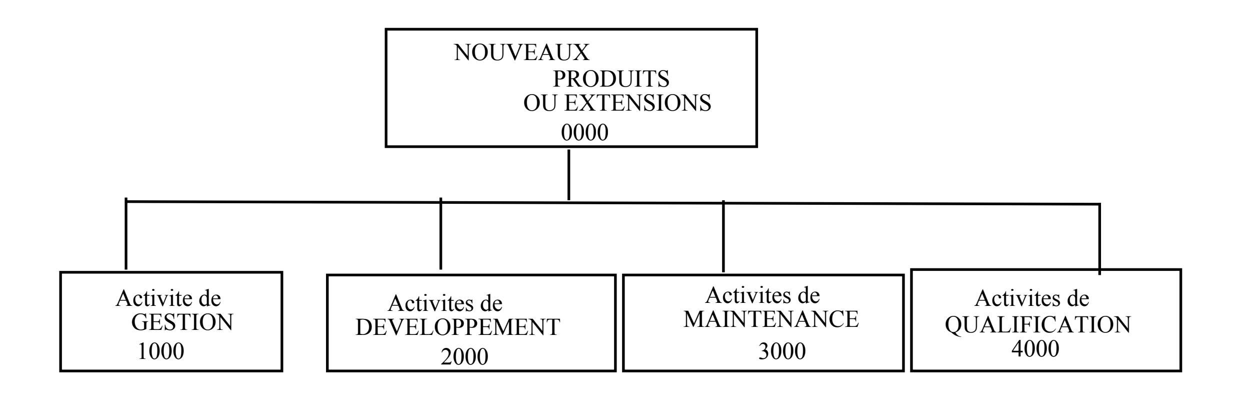
jusqu'à obtenir des activités qui soient bien définies et facile à gérer c'est à dire dont les entrées et résultats sont parfaitement identifiés et dont la responsabilité est confiée à une ou des personnes précise(s).

Elle permet au chef de projet de planifier son projet en établissant le graphe *PERT* [1] de celui ci Elle permet le suivi budgétaire du projet en liaison avec les activités élémentaires identifiées lors de la construction du PERT

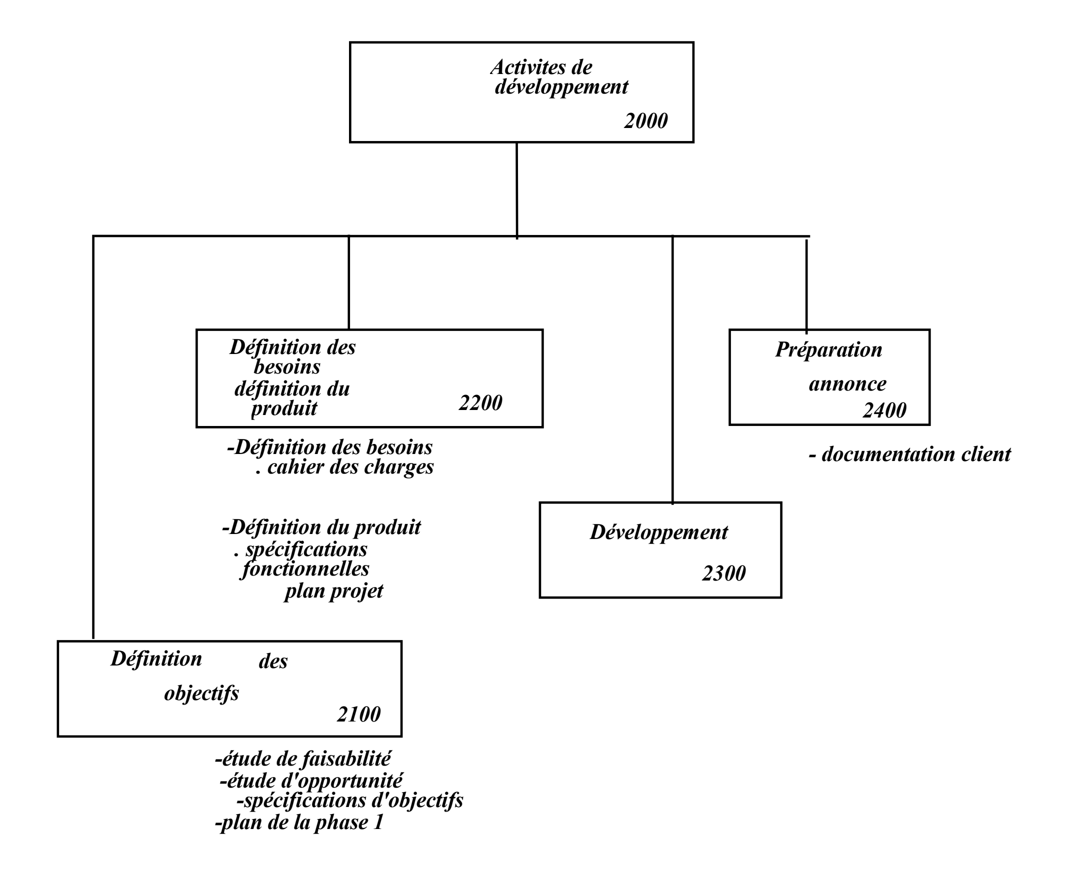
La WBS doit être complète car elle conditionne l'élaboration du PERT et donc du budget. Elle doit être non ambiguë dans la définition des activités Elle doit définir des activités dont le résultat est mesurable, ces activités feront l'objet d'affectation de ressources. Il est à noter que certaines activités existent dans tout projet:

- Élaboration des différents documents du cycle de vie
- Inspections, - Revues
- Construction d'outils
- Apprentissage

Exemples de schémas de WBS, arborescences



[1] décrit plus loin

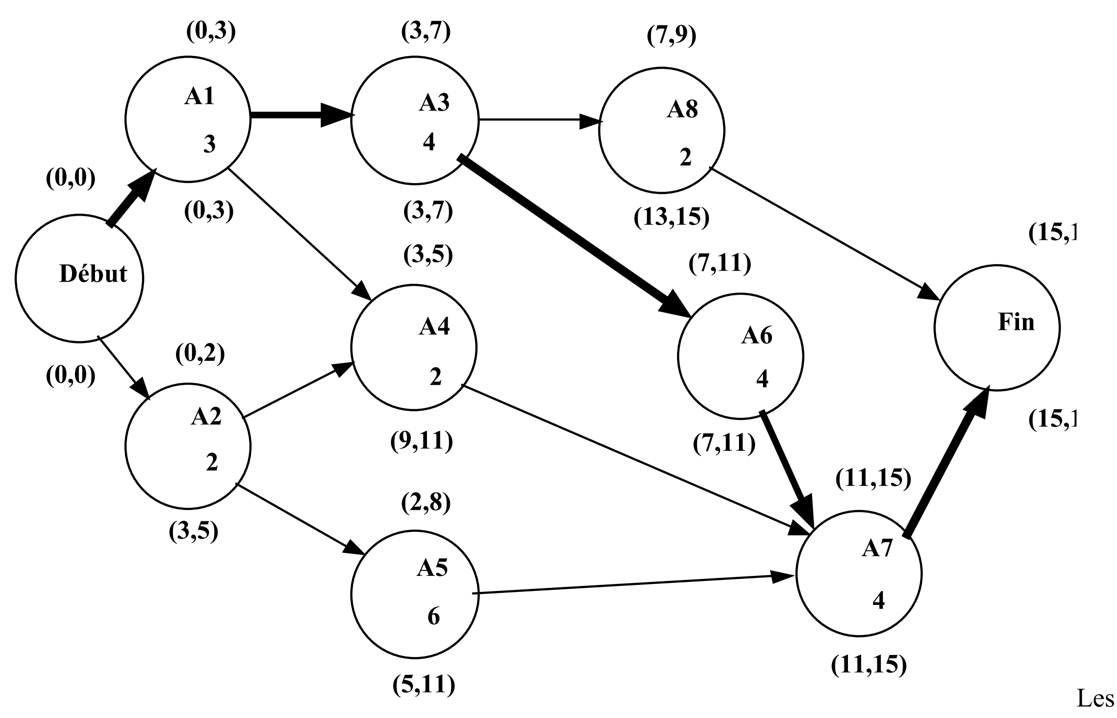


3.4.2. Ordonnancement et dépendances: Graphe PERT

On utilise un graphe de dépendances: (PERT: Project Évaluation and Review Technique). Pour chaque tâche, on indique une date de début et de fin au plus tôt et au plus tard.

Le diagramme permet de déterminer **le chemin critique** qui conditionne la **durée minimale du projet**. Ces techniques ne sont en aucun cas propres au génie logiciel; elles sont par exemple très fortement appliquées dans le BTP.

Exemple:



Les durées apparaissent dans les cercles,

les couples au dessus sont les dates de début et de fin au plus tôt,

les couples au dessous sont les dates de début et de fin au plus tard.

A ce niveau les activités sont les plus élémentaires possibles

Si le projet nécessite plusieurs équipes, on a des PERT à plusieurs niveaux

3.4.3. Répartition des activités: diagramme de Gantt

Ce diagramme permet de faire apparaître la répartition des activités dans le temps et l'affectation des individus. Il est indispensable pour définir le plan projet. Il fournit une description détaillée des coûts (en homme x mois) et des dates pour chaque *tâche* et pour chaque *phase* du projet .

A chaque tache/sous tache on associe un **objectif** qui permet de repérer la **terminaison** de l'activité. On définit des points clés (milestones) qui servent de borne intermédiaire (exemple: réalisation d'un prototype)

On définit les revues qui sont aussi des milestones, on n'oubliera pas la tâche de préparation de la revue.

activité	1	2	3	4	5	6	7	8	9	10	Cumul act.
1	A	A,B									3
2.1			A,B								2
2.2				A,C	A,C						4
2.3				B	B,D	B,D					5
2.4						A,C	C				3
2.5								C,D			2
3									C,D	D	3
Tot.mens	1	2	2	3	4	4	1	2	2	1	
Cumul	1	3	5	8	12	16	17	19	21	22	

Attention aux pics brutaux, le passage du mois 6 au mois 7 n'est pas judicieux, on passe de 4 à 1; il est souvent nécessaire de procéder à un lissage.

3.5. PLAN PROJET

C'est l'un des éléments clés du cycle de vie. C'est aussi un élément de base de la planification du développement

Le responsable du plan projet est le chef de projet du développement

L'élaboration du plan projet se fait dès la phase de planification. On procède ensuite par actualisations successives et raffinements.

3.5.1. Contenu du plan projet

Le plan projet contient les éléments qui permettent de définir le projet

- Domaine d'application
  - Objectifs
  - Principales fonctionnalités
  - Autres caractéristiques
    - Les limites
    - Les performances
- Scénario de développement
  - Classification des priorités et des contraintes
  - Brève description de chaque composant
- Ressources
  - Humaines
  - Matérielles
  - Logicielles
- Coût
- Calendrier

Ce document n'est pas forcément très long. C'est avant tout un instrument de travail pour le chef de projet et son management. C'est une garantie de qualité pour le client.

3.5.2. Modèle de plan projet IEEE (1987)

Introduction
- résumé du projet
- fournitures (avec les dates, documents, codes, jeux de tests)
- procédures pour faire évoluer le plan projet
- références des documents cités dans le plan projet
- définitions et acronymes

Organisation du projet
- modèle de processus (organisation, bornes intermédiaires, revues avec participants but matériel dates..)
- organisation structurelle (voir plus loin organisation des équipes)
- limites et interfaces (description des interactions avec autres projets...)
- rôles et responsabilités

Management
- Objectifs et priorités
- Hypothèses dépendances contraintes
- Gestion du risque
- moyens de contrôle (rapport d'activités...)
- ressources humaines (équipes affectées au projet)

Techniques
- méthodes outils techniques employés (ex: OMT, OMTools, C++...)
- documentation (dates de remise de la doc..)
- fonctions support (gestion de configuration, tests, assurance qualité..)

Calendrier, budget, lots
- lots (avec le détail des activités et tâches constituant chaque lot)
- dépendances (matérielles et logicielles pour le projet mais aussi pour tests, simulation...)
- ressources (autres qu'humaines, Identification des fournisseurs, sous-contractants, …)
- budget et allocation de ressource
- calendrier détaillé (Gantt, comprenant les dates de revues)

Risques Problèmes Préoccupations
- Il s'agit de lister ici les problèmes potentiels que l'on pressent. En particulier
- Hypothèses sur les effectifs.
- Difficultés techniques (nouvelle technologie, …)
- Conflits sur le calendrier

3.5.3. Planification pour le paradigme orienté objets

Les méthodes d'évaluation vues plus haut s'appliquent assez bien dans le cas de projet réalisés en suivant le paradigme objet. La forte modularité induit une évaluation plus facile de chaque élément. Il faut bien évidemment tenir compte lors de l'évaluation de l'ensemble des interactions existant entre chaque composant qui génèrent un overhead qu'il ne faut pas négliger.

La méthode COCOMO peut être appliquée pratiquement telle quelle (si ce n'est quelques changements mineurs dans certains coefficients) pourvu qu'il n'y ait pas trop de réutilisation. Si la réutilisation intervient pendant le développement, clairement les coûts et durées devraient être réduits. Il n'existe aujourd'hui aucun outil d'évaluation fiable prenant en compte la réutilisation.

Par contre si le développement se fait en vue de la réutilisation il faudra prévoir une augmentation des estimations. Des publications récentes ont montré que le développement de composant réutilisable peut prendre jusqu'à 3 fois plus de temps que pour un composant similaire non réutilisable.

On peut supposer qu'a long terme les économies faites grâce à la réutilisation surpasseront le surcoût occasionné (à rapprocher de l'amortissement d'un investissement)

3.6. SUIVI DE PROJET

Une fois le planning défini il importe de le respecter. Au niveau d'équipes de 4 à 10 personnes, le responsable de projet utilise des fiches de suivi (ou rapports d'activités) qui permettent d'identifier les dérives et d'assurer le contrôle budgétaire. Il en déduit :

- L'actualisation du graphe *PERT*
- Le diagramme de GANTT actualisé
- Des diagrammes à 45° qui donnent une perception visuelle de la dérive par rapport aux objectifs.

3.6.1 Rapports d'activités

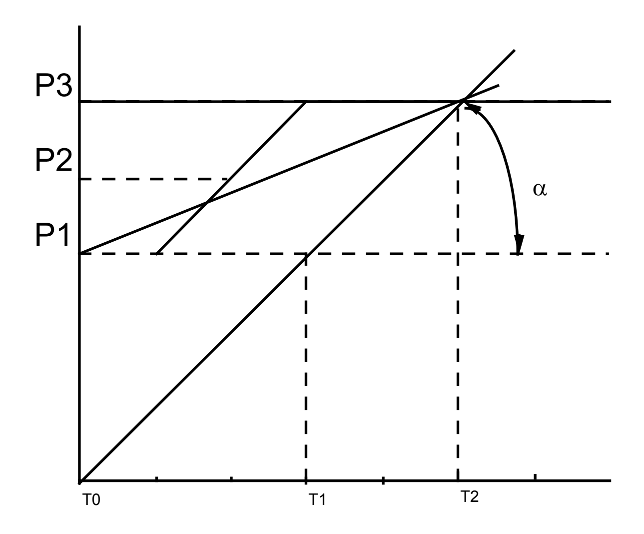
Chaque membre de l'équipe fournit un rapport d 'activité hebdomadaire succinct qui décrit

- l'objectif à atteindre pour la semaine
- le temps passé sur les différentes tâches
- si les objectifs ont été atteints et si ce n'est pas le cas pour quelle raison.

A partir des rapports individuels, le chef de projet peut établir un rapport d'activité de l'équipe.

3.6.2. Diagrammes à 45°

Ils permettent de mesurer l'ange de dérive des objectifs. En ordonnée on représente les points clé (exemple revue de fin de phase, obtention du prototype) tels qu'ils sont planifiés à T0. On utilise la même échelle en abscisse et en ordonnée, d'où l'appellation 45° On suppose que les abscisses représentent des mois. Sur la figure suivante, la revue P1 est prévue à T0+3 (=P1)



A l'instant T0+1 la revue est toujours prévue à T=T0+3
A l'instant T0+2, la revue est repoussée d'un mois (P2)
A l'instant T0+3 (T1) la revue est encore repoussée d'un mois (P3)
A l'instant T1+1, la revue est maintenue à P3
A l'instant T2=P3, la revue a effectivement lieu.

On mesure l'angle α formé par la droite passant par les points de première et dernière estimation horizontale, si a =0 alors l'objectif initial a été atteint.

On établit des seuils acceptables,

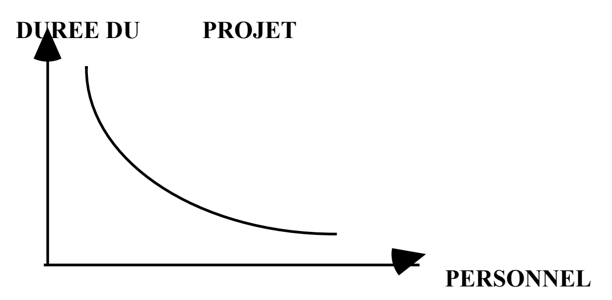
1. 20° bon niveau

2. < α < 30° niveau moyen
α > 30° niveau médiocre, dérive importante

3.7. ORGANISATION DU TRAVAIL

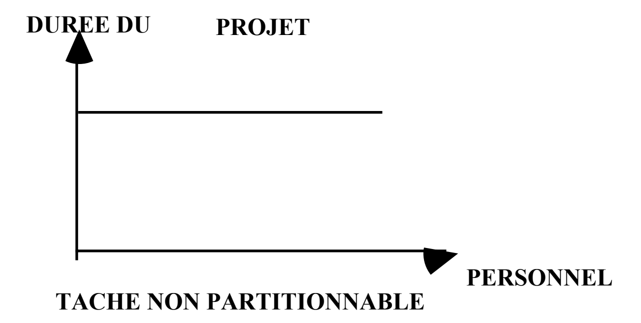
3.7.1. Éléments de réflexion pour le partage du travail

La répartition du travail entre les membres d'une équipe ne peut pas toujours se faire. Cela dépend de la nature de la tâche à effectuer; Les diagrammes ci-dessous donnent une idée de la durée du projet en fonction du nombre de personnes affectées dans 3 cas différents.



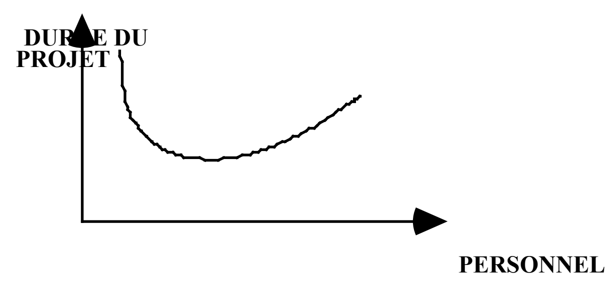
TACHE PARFAITEMENT PARTITIONNEE

Exemple: codage et tests unitaires



TACHE NON PARTITIONNABLE

Exemple analyse et conception globale



TACHE NECESSITANT UNE COMMUNICATION COMPLEXE

Exemple développement d'un système d'exploitation embarqué avec fortes contraintes espace-temps

3.7.2. Les acteurs principaux d'un projet

Le chef de projet

Il est responsable de l'ensemble du projet, à la fois au niveau des coûts et des délais. Il est responsable de la rédaction et du suivi du plan projet.

Le responsable qualité

Il est responsable de la mise en oeuvre du manuel qualité sur un projet donné. Il rédige et fait appliquer le plan qualité du projet.

Le responsable des ressources matérielles

Il assure la disponibilité du matériel conformément à la planification.

Le responsable de l'intégration

Il est responsable de la mise en oeuvre du plan d'intégration

Le responsable de la qualification

Il est responsable de la mise en oeuvre du plan de qualification

Le responsable des performances

Il est responsable des tests de performances conformément au cahier des charges et au plan qualité. Il prend les mesures nécessaires pour atteindre les objectifs de performance définis dans le cahier des charges (simulation, prototypes)

Le responsable de la maintenance

C'est le responsable de l'équipe qui prendra en charge la maintenance du produit (en général différente de l'équipe de développement).

Le responsable de la documentation

Il est responsable de la documentation du projet. Ceci ne veut pas dire qu'il rédige toute la documentation associée au projet, mais plutôt qu'il s'assure de sa rédaction. Il définit les normes de présentation des documents en accord avec le manuel qualité de l'entreprise. Il s'assure de la mise à jour de la documentation.

REMARQUE:

Sur les gros projets, le chef de projet pourra nommer des responsables pour les phases amont du cycle de vie (analyse des besoins, spécifications, conception, codage).

3.7.3. Profil des membres d'une équipe

Ces profils doivent être choisis en fonction de l'intervention dans les différentes phases du cycle de vie. Nous répertorions les qualités requises suivant les phases.

Définition et analyse des besoins

Savoir anticiper
Savoir analyser les stratégies

Spécifications fonctionnelles

Clarté, précision, cohérence, rigueur
L'idéal serait un ingénieur ayant la double compétence: utilisateur et concepteur

Conception

Cette phase demande des communications intenses.
Capacités à formaliser et à abstraire

Programmation et tests unitaires

Discipline de programmation (style), rigueur, communication, sens du groupe

Intégration

C'est une phase où la compétence requise est bien souvent celle d'un ingénieur système.

Qualification

Ici il s'agit plutôt de la compétence d'un ingénieur d'application, c'est à dire très proche du domaine du produit à qualifier

Maintenance

La maintenance requiert des qualités de rigueur, analyse et expertise

3.7.4. Nécessité de la structuration

Dans un groupe non structuré de N personnes il y a N * (N -1) / 2 interactions "I"

Exemple

N = 3 --> I = 3

N = 11 --> I = 55

Il faut donc structurer les équipes pour diminuer le temps passé à la communication.

Calcul du temps en heures/jour pour des groupes non structurés

N = 4 --> T = 2

N = 8 --> T = 4

N = 16 --> T = 6

N = 24 --> T = 7.5 !!

Cependant la communication

- Améliore la compréhension du projet.
- Permet une plus grande mobilité dans le projet.

Mais aussi

- Fait perdre du temps.
- Peut nuire a la documentation

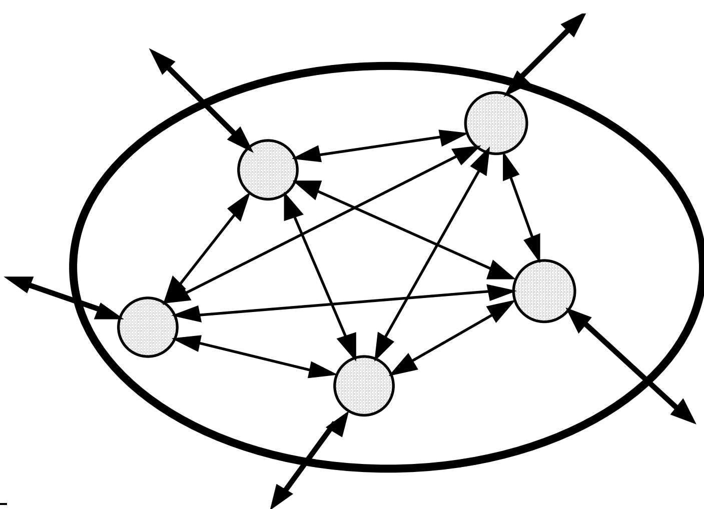
(dont on peut croire qu'elle devient moins nécessaire!).

3.7.5. Les types d'organisation

Les extrêmes:

PETIT GROUPE DE TRAVAIL SANS AUTORITE DEFINIE : EGOLESS PROGRAMMING

Le travail s'effectue par consensus; le travail de chacun devient le travail de tous. L'équipe s'enrichit par le travail quotidien en commun, les revues et inspections de code. On élimine l'attachement à *son* programme, *ses* idées.



-

Cette notion apparaît pour la première fois dans [Hein 71]. Elle est reprise dans les techniques de revues et d'inspection de code développées au chapitre validation/vérification.

L'ORGANISATION "CHIEF PROGRAMMER TEAM"

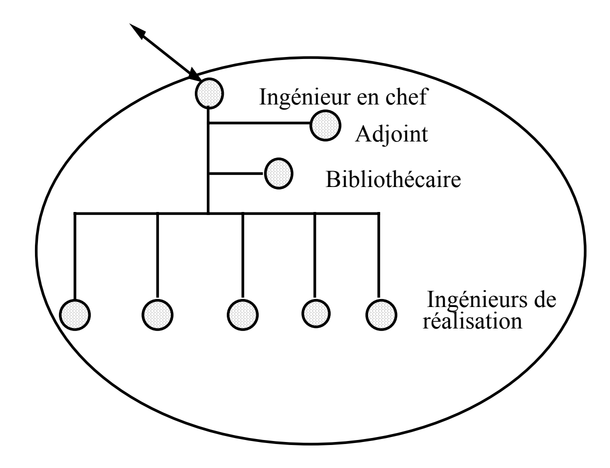
Il s'agit d'une structuration de l'egoless programming, faisant suite à un projet réussi publié par IBM en 1970.

Un ingénieur en chef ou chef de projet ou senior programmeur dirige l'équipe composée de 2 à 5 personnes. Il coordonne, planifie et vérifie le travail effectué. L'adjoint seconde l'ingénieur en chef et peut à tout instant le remplacer.

L'équipe peut s'adjoindre les services d'un ou plusieurs experts (télécoms, bases de données...).

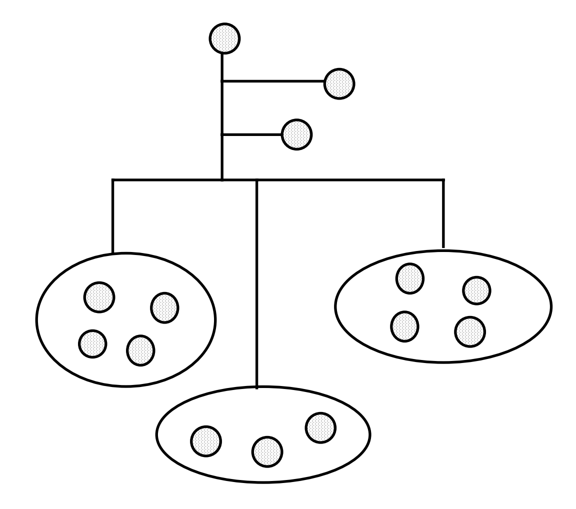
Le bibliothécaire est une ressource partagée par divers projets. Son importance ne peut être négligée. Il maintient et contrôle les éléments de la bibliothèque de programmes, vérifie l'intégrité des configurations (sources, données, bandes, documentation...) Il collecte les données qui permettront d'évaluer la productivité. L'amélioration de la qualité passe par cette activité d'archivage et de contrôle.

Chacun a accès au travail de tous, l'ingénieur en chef a le leadership technique.

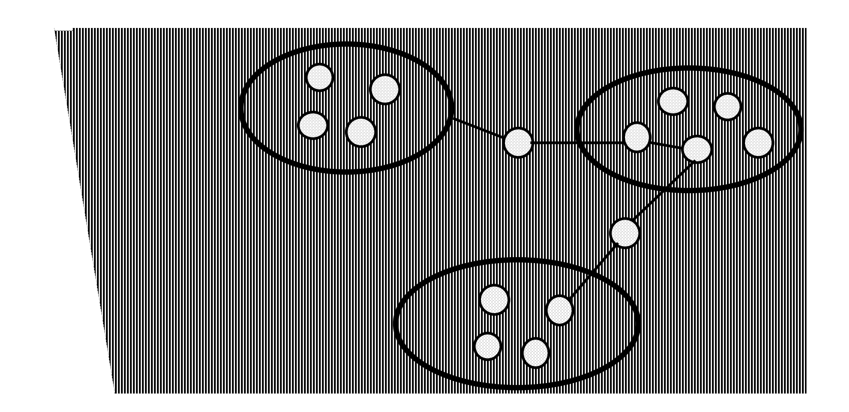


Les structures intermédiaires:

CHEF DE PROJET POUR PLUSIEURS EQUIPES



COMITÉ DE DIRECTION ET PLUSIEURS ÉQUIPES:



3.7.6. La taille des équipes

Structuration en groupes de 2 à 10 personnes
Nombre magique: = 7 ± 2

3.7.7. Facteurs humains

Outre l'organisation du travail d'équipe il faut veiller

- Aux motivations individuelles et à la motivation collective de l'équipe.
- Aux relations entre les membres et avec l'extérieur.
- A la dynamique du chef de projet.
- Aux écueils à éviter :
  - Sur-spécialisation
  - Trop de niveaux
  - Pas assez de niveaux
  - Déresponsabilisation
- A la formation et en particulier au temps d'apprentissage du domaine.

Mais il faut savoir que de grandes variations sont liées aux facteurs humains

- Temps de codage	1 A 25
- Temps de mise au point	1 A 26
- Temps CPU utilisé	1 A 11
- Temps d'exécution	1 A 13
- Lignes écrites	1 A 5
- Nombres d'erreurs	1 A 10

Ces variations *peuvent être réduites* par

- Méthodologies de développement.
- Standards, normes, …
- "Confort" : matériel, logiciel

en aucun cas elles ne peuvent être annulées

3.8. ORGANISATION DE LA DOCUMENTATION

La documentation associée à un projet doit être le reflet de la vie de ce projet. Elle est élaborée et mise à jour tout au long du projet. Elle est constituée par un ensemble de documents dont certains sont associés aux phases "verticales " du cycle de vie et d'autres sont plus transversaux.

Elle est réalisée par des spécialistes à partir de documents fournis par l'équipe de développement

3.8.1. Documents de référence

Il s'agit des diverses documentations associées au projet, on distingue les éléments destinés aux utilisateurs de eux destinés aux développeurs pendant le développement et aux mainteneurs en phase de maintenance.

Documents liés à une phase du cycle de vie (essentiellement destiné aux développeurs et mainteneurs)
- cahier des charges
- document de spécifications fonctionnelles
- plan d'intégration
- manuel de conception globale et détaillée
- procédures de validation
    tests unitaires, d'intégration, de validation, de non régression
- code

Documents transversaux à la vie du projet
- Le glossaire, la liste des abréviations.
- Le plan projet
- Le plan qualité

Documents remis au client
- Le manuel utilisateur
- Le manuel d'installation
- Le manuel de maintenance

3.8.2. Forme des documents

La documentation adopte une forme spécifiée par le plan qualité. Elle est conforme aux normes et conventions édictées par le manuel qualité de l'entreprise. Elle doit de toute façon comporter les notions de numérotation, chapitre, statut, date, classification , mots clés. Tout comme le reste du produit, la *documentation doit être maintenue* pour rester en phases avec les évolutions du produit.

NUMÉROTATION: Elle peut être unique ou par classe ou par tâche

STATUT :

Le statut peut prendre 4 valeurs
D définitif et approuvé
P partiel, sujet à modification et /ou ajouts
R révisé (actualisé)
I incomplet (et dans ce cas la date à laquelle le chapitre doit être complet)

Exemples

révision	(165 révision 3)
question	(165 question 2)
commentaire	(165 commentaire 5)

CLASSIFICATION
document de travail
spécification
conception
code
test
administration
compte rendu de réunion

MOTS CLÉS : Ils permettent des recherches automatiques

CONTENU : Le contenu sera précisé dans les chapitres associés aux différentes phases du cycle de vie.

RÈGLES DE STYLE
Utiliser des formes actives
Faire des phrases courtes, une seule idée par phrase
Faire des paragraphes courts (7 phrases au plus)
Préférer les listes aux phrases
Ne pas hésiter à répéter si nécessaire
Pas de verbiage
Éviter les références du style 1.2.3.4.5 pour parler d'un objet, plutôt nommer l'objet
Être précis, définir les termes (glossaire)
Attention à la grammaire et l'orthographe

3.8.3. Documentation utilisateurs

Manuel utilisateur

Il doit donner une impression initiale correcte:
Ce n'est pas une plaquette publicitaire
Il doit être possible de lire sans aller dans tous les niveaux de détail

Structure possible:
- Description fonctionnelle simple à l'aide d'exemples de ce que le système peut faire
- Document d'installation: décrit la procédure d'installation, souvent assorti d'un fichier A LIRE (READ ME)
- Manuel d'Introduction (utilisation "normale", exemple simples), classé par rubriques de difficultés croissantes
- Manuel de Référence: contient toutes les références aux possibilités du système classées par ordre alphabétique.

Autres documents destinés aux utilisateurs
Plaquette publicitaire
Carte de Référence
Aide en ligne (man)

3.8.4. Documentation interne

Nous verrons la description précise de ces documents dans les chapitres concernés. Nous insistons tout particulièrement sur la nécessité de maintenir la cohérence de tous les documents .

Pour cela il est souhaitable de gérer TOUS les documents avec un gestionnaire de versions, par exemple SSCS, RCS, MAKE...

L'utilisation d'un dictionnaire des données permettra d'avoir des détails
- sur toutes les entités manipulées
- avec des références croisées

3.9. CONCLUSION

La gestion de projet est une des meilleures garanties de l'assurance qualité, elle permet la maîtrise du processus de développement. Il existe un certain nombre d'outils pour aider le chef de projet dans sa tâche.

Les outils manipulant des graphes PERT et des diagrammes Gantt: Mac Project, Method1, Teamwork Access.

Les méthodes COCOMO et Function points peuvent être programmées très simplement sur des tableurs.

Enfin pour la documentation technique des outils comme Interleaf ou Framemaker seront préférés à Word pour leur facilité d'interfaçage avec les outils d'analyse et conception qui seront vus au chapitre suivant.

3.10 REFERENCES

de "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0" , Bohem et al. 1995)

http://www.ifpug.org/

McConnell, Steve, Rapid Development,Microsoft Press,1996 Presents all the factor to achieve rapid development, from risk evaluation, good practices, classical mistake, etc ... to team psychology or negociating. Really great.

Boehm, Barry W., Software Engineering Economics,Englewood Cliffs N.J.: Prentice Hall 1981 COCOMO cost-estimation model, by its creator.

DeMarco, Tom, Controlling Software Projects, New York: Yourdon Press,1982. Describes several estimation models.

Putnam,Lawrence H., and Ware Myers. Measures of Excellence: Reliable Software on Time, Within Budget. Englewood Cliffs N.J.:Yourdon Press,1992. Presents a full-fleged software-project estimation.Explains how to calibrate a simple cost-estimation model to your organisation and how to use it to estimate medium to large projects.

Jones,Capers. Assessment and Control of Software Risks. Englewood Cliffs N.J.:Yourdon Press,1994. Estimation,Project management.

Gilb,Tom. Principles of Software Engineering Management. Workingham,Englang: Addison Wesley,1988. Practical advices for estimating software schedule.Focus on the importance of controlling the project to achieve your objectives rather than passive prediction about it.

Dreger,Brian. Function Point Analysis,Englewood Cliffs N.J.: Prentice Hall 1989

Function Point Analysis. Jones,Capers. Applied Software Measurement:Assuring Productivity and Quality, New York:McGraw-Hill,1991. Function Point Analysis.

Boehm, Barry W., Software Engineering Economics, (1981) Cocomo dans toute sa profondeur, par l'auteur de Cocomo lui même.

Boehm, Barry W., Software engineering economics, IEEE Trans. on Software Engineering.,(1984) Détails pour assigner les valeurs aux 15 facteurs de productivité.

Shepperd, Martin, Fundation of Software Measurement, (1994) Le pourquoi et comment des différentes méthodes de mesure de logiciels.

Katwijk, J. van, Inleiding software engineering, (1994)

Pressman, Roger S., (adapted by Darrel Ince) Software engineering "A practitioner's approach", (1994)

Universite de Californie du Sud, Centre de développement de logiciel

C'est l'endroit ou COCOMO a été développé et continu de l'être. L'état du développement de COCOMO II. Les sponsors du programme de recherche Télécharger des logiciels et de la documentation COCOTS Projet de recherche pour estimer les coûts, les efforts et planifier des projets logiciels utilisant des composants existants.

COQUALMO Un modèle d'estimation logiciel pour équilibrer coût, planning et qualité. CORADMO Le modèle COCOMO RAD est une extension de COCOMOII centrée sur les coûts de développement logiciel utilisant les techniques du 'Rapid Application Development'.

Un projet estimé avec COCOMO 2.0

Bibliographie Quelques estimateurs en ligne Expert COCOMO II par USC avec estimation de risque Par Dr. Ray Madachy (Implementation très innovante donnant également une évaluation des risques. Interface graphique conviviale.)

COCOMO II par USC (Interface lente et moins conviviale.)

Predicate logic software systems Estimation en ligne du modèle intermédiaire de COCOMO 81. COCOMO Intermédiaire Interactive Développé par 2 étudiants de l'Université de Caroline du Nord. Interface graphique très sympathique. Manuel d'utilisation et Information sur le calculation (COCOMO 81).

La NASA Estimateur COCOMO : modèle de Base (COCOMO 81). COCOMO II COSTAR Entreprise commercialisant COCOMO II.

Critique de COCOMO Par des consultants spécialisés de la gestion du risque, par l'utilisation des réseaux Bayesian (COCOMO II étant basé sur l'approche Bayesian). Modèles d'évaluation de projet logiciel

Les modèles orientés activités Par M. JM Jaeger **EN FRANCAIS!!**

Université du Texas : Projet COCOMO Présentation de l'histoire de COCOMO. (Manuel utilisateur et Estimateur en ligne (COCOMO 81)

Les buts de COCOMO Explications d'un professeur de l'Université Britannique.

Estimation des coûts logiciels Université de Calgary. Bon résumé de la problematique même si il ne parle pas directement de COCOMO.

Methodes d'estimation des coûts logiciels Présentation rapide des différentes méthodes d'estimation, suivie par une présentation assez détaillée des 3 modèles de COCOMO 81. Formules COCOMO Présentation intéressante en diagrammes commentés.(COCOMO 81) COCOMO : équation de base (COCOMO 81), Université Thaïlandaise

. Bibliographie d'Estimations de Logiciel Compilé par l'Université de Bournemouth, UK